

Genetic Evolution of Shape-Altering Programs for Supersonic Aerodynamics

Robert A. Kennelly, Jr.

High Speed Aerodynamics Branch, MS 227-6
NASA Ames Research Center
Moffett Field, CA 94035-1000
RAKennelly@mail.arc.nasa.gov

ABSTRACT

Two constrained shape optimization problems relevant to aerodynamics are solved by genetic programming, in which a population of computer programs evolves automatically under pressure of fitness-driven reproduction and genetic crossover. Known optimal solutions are recovered using a small, naive set of elementary operations. Effectiveness is improved through use of automatically defined functions, especially when one of them is capable of a variable number of iterations, even though the test problems lack obvious exploitable regularities. An attempt at evolving new elementary operations was only partially successful.

1. Introduction

Optimal aerodynamic shaping offers a wide variety of problems ranging from two-dimensional airfoils in linearized (inviscid) sub- or supersonic flow to three-dimensional arrangements of interacting components in a flow field dominated by viscous effects. While some of the problems can be attacked analytically, the most complex configurations are still "optimized" by cut-and-try methods, with expensive and slow experimental validation of each step taken. For cases of intermediate difficulty, gradient-based numerical optimization is becoming practical for well-structured problems employing well wrung-out aerodynamic analysis codes. Indeed, such optimization may be crucial for the development of future supersonic transport aircraft. But appropriate parameterization remains something of an art, and the stringent requirements of this technique for reliability and precision are difficult to meet without violating a third requirement: sufficient speed to permit hundreds or thousands of objective function evaluations in a "reasonable" length of time.

Because the marketplace for air vehicles is evolving concurrently with design methods, each design problem tends to be different. Powerful (but complex) methods for solving last year's design problem may not be the best means to realize the shorter design cycles (Rubbett 1994) felt to be critical to commercial success. The work described herein, an examination of genetic programming

(Koza 1992) in the context of shape optimization, represents a small step in the direction of a more general class of aerodynamic design optimization techniques. The long-term goal is to investigate whether the new approach's advantages of inherent flexibility and tolerance of low-precision performance measures are sufficient to overcome its relative lack of efficiency and uncertain convergence properties.

Toward this end, two simple, two-dimensional shape optimization problems, one purely geometric and the other based on supersonic aerodynamics, are solved using genetic programming (GP) to evolve a population of LISP-like "programs." These objects, appropriately interpreted, direct a sequence of alterations of the initial shape. The evolution of the individuals in the program population proceeds under the influence of Darwinian fitness-driven reproduction, modified in the majority of instances by the genetic "crossover" operation (see, for example, Holland 1992 or Koza 1992). If one of the individuals succeeds in satisfying a termination criterion, i.e., the shape resulting from the program-directed modifications meets the specified requirements, the problem is considered solved.

The experiments to be described are exploratory, and there is no assurance that genetic programming will ever lead to the flexibility sought. These samples of shape optimization may, however, provide signposts for future, more general work on structural requirements of the genetic programs, sufficiency of the elementary operation set, and appropriate choices for the basic parameters of the GP process.

Related work is briefly reviewed in Section 2, followed in Section 3 by a description of the test problems. Section 4 presents several methods of solution, and the results follow in Section 5. Possibilities for further research are the topic of Section 6, and the paper is summarized in Section 7.

2. Background

Aerodynamic shape optimization, understood in what follows to refer to numerical optimization of the detailed shape of an aircraft's external surface, has a history of at least three decades. Some early research results are provided in the list of references (Hague, Rozendaal, and Woodward 1968; Hicks, Murman, and Vanderplaats 1974). These, and other studies, all employ perturbations to a given initial configuration, often subject to explicit or implicit constraints. The optimization algorithm iteratively modifies parameters representing the perturbations as it seeks improvements in some computed, scalar measure of

to {1, 2, ..., 15} to remain within the array bounds. In addition, most of the problem formulations used a small set of elementary shape-modifying operators, along with one or more automatically defined functions (ADF0, ADF1, ...) which are themselves composed of the elementary operators. These operators were BUMP, which multiplies $y_{hotSpot}$ by 1.01, and GRIND, which, for $2 \leq hotSpot \leq 14$, replaces $y_{hotSpot}$ by $(y_{hotSpot+1} + y_{hotSpot-1})/2$, the mean of the two neighboring points. An additional terminal, TAP, was used for some of the later runs. It modifies the shape as follows:

$$y_{hotSpot} = y_{hotSpot} - 0.010 * MIN_THICK;$$

the latter quantity is the thickness constraint value for the airfoil problem. All of these terminals return a value of 1.0, unconditionally.

BUMP and GRIND were naive choices in that they were simply the first plausible operations dreamed up—while selected for their presumed utility in modifying a pre-existing shape, they have no special characteristics suiting them to the two problems posed. TAP, as will be seen, was chosen in response to observed behavior on the airfoil problem. But all three operations can be viewed as intermediate in functionality: they produce modest, local changes in shape, in contrast to the higher order “tools” such as will be seen to evolve as automatically defined functions. At the other extreme was a more-nearly “hands-off” approach attempted with the semicircle problem, where the only operations were $\{+, -, *, \%\}$. This latter experiment will be described more fully below.

The function set consisted of PROGN, a 2-branch “glue” function which merely evaluates both of its arguments, returning the value of the last, and IPB, an iteration-performing function. In many of the later runs, an ephemeral random constant, CONST, is also available to provide an iteration count for IPB. The value of each instance of CONST is determined once, chosen from the floating point “integers” 1.0 to 15.0, at the time the initial population is created. The effect pointer, $hotSpot$, is

arbitrarily reset to one at the beginning of each iteration performed by IPB.

The genetic software engine employed was “DGPC—Dave’s Genetic Programming in C”. (Andre, David, 1995. <http://www.cs.berkeley.edu/~dandre>). Using hooks provided by the DGPC software, the problem-specific terminals and functions were coded in C. A modification to the section of DGPC which evaluates ADF branches turned a specific ADF into the function IPB, controlled by the value of its input as described below. Another modification was required to permit each ADF, upon termination, to modify $y[hotSpot]$ as a side-effect—this was part of an attempt to evolve the form of the elementary operations rather than relying on BUMP, GRIND, and TAP. The program was compiled and run on various models of Indigo² workstation (Silicon Graphics, Inc., Mountain View, CA).

5. Results

5.1. Semicircle

Of a number of variations on choice of terminals, structure, and run parameters, four have been selected for discussion, to be followed by brief mention of some dead ends.

5.1.1 RPB with Two Specialized ADFs

This structure was motivated by the idea that interesting meta-tools might evolve by requiring the re-use of two specialized automatically defined functions; no other means of changing the shape was possible. While solutions were obtained, it must be confessed that the resulting ADFs have so far defied simple interpretation.

Following Koza (1992), we describe the preparation for solution of the problem by GP in terms of five major steps:

1. The terminals consist of the navigation operators INC and DEC, two elementary shape operations BUMP and GRIND, and two automatically defined functions ADF0 and ADF1, which despite their names actually serve as terminals rather than functions in this case. The ADFs

Table 1. GP tableau for the first version of the semicircle problem, “RPB with Two Specialized ADFs.”

Objective	Minimize perimeter subject to area constraint.	
Terminals	RPB: ADF0, ADF1 ADF0: INC, DEC, BUMP ADF1: INC, DEC, GRIND	
Functions	RPB: PROGN (2-branch) ADF0: PROGN ADF1: PROGN	
Standardized Fitness	$(\text{Perimeter} - \pi/2) + 1,000 \max(\pi/8 - \text{Area}, 0)$	
Success Predicate	Standardized Fitness ≤ 0.010	
Parameters	Population 4,000 Generations 31	Max Nodes Per RPB 200 Max Depth New Tree 5

Table 2. GP tableau for the "Hands Off" version of the semicircle problem.

Objective	Minimize perimeter subject to area constraint.	
Terminals	RPB: INC, DEC, ADF0, ADF1 ADF0: Y, Y+, Y-, CONST (values from 1 to 15) ADF1: Y, Y+, Y-, CONST (values from 1 to 15)	
Functions	RPB: PROGN (2-branch) ADF0: +, -, *, % ADF1: +, -, *, %	
Standardized Fitness	$(\text{Perimeter} - \pi/2) + 1,000 \max(\pi/8 - \text{Area}, 0); \leq 1.0e+30$	
Success Predicate	Standardized Fitness ≤ 0.010	
Parameters	Population 40,000 Generations 101	Max Nodes Per RPB 1,000 Max Depth New Tree 7

always return 1.0. The RPB was otherwise unchanged: shape modifications could only be performed indirectly, by calling either ADF or IPB. All of the GP run parameters were restored to the values in Table 1.

Twenty straight successes were recorded. The average run required 10 generations and lasted 3.5 minutes. One individual is positively svelte by comparison to the results of the previous runs, even with the extraneous ARG0 terminals in the IPB ("ADF_Num 1", in boldface, below) supplied by DGPC but unused here:

Run 497 Gen:5 BestFitness:0.00792

RPB_Num 0 (17 nodes)

```
(ADF1 (PROGN (ADF1 (ADF1 5.0)) (PROGN
(PROGN (ADF1 (ADF1 (ADF1 (ADF0))))
(PROGN (ADF1 7.0) (ADF0))) (ADF1 5.0))))
```

ADF_Num 0 (31 nodes)

```
(PROGN (PROGN (PROGN (PROGN (BUMP) (INC))
(PROGN (INC) (GRIND))) (PROGN (PROGN
(GRIND) (DEC)) (PROGN (INC) (BUMP))))
(PROGN (PROGN (PROGN (BUMP) (GRIND))
(PROGN (BUMP) (DEC))) (PROGN (PROGN
(GRIND) (DEC)) (PROGN (INC) (BUMP))))
```

ADF_Num 1 (31 nodes)

```
(PROGN (PROGN (PROGN (PROGN (DEC) (ARG0))
(PROGN (ARG0) (BUMP))) (PROGN (PROGN
(BUMP) (ARG0)) (PROGN (ADF0) (GRIND))))
(PROGN (PROGN (PROGN (ADF0) (GRIND)) (PROGN
(ADF0) (INC))) (PROGN (PROGN (ARG0) (BUMP))
(PROGN (ADF0) (INC))))
```

The formal structure of the GP with an IPB will be clarified below, in Table 3, where a tableau for the parameters of the airfoil problem is presented.

5.1.4 RPB with Two ADFs; "Hands Off"

We briefly describe an attempt to progress from evolution of tool-using behavior to evolution of the tools themselves. This version of the problem is similar in structure to the case considered in Section 5.1.2, *RPB with No ADFs*, except that where in the previous version the RPB was permitted to use BUMP and GRIND, in the present case the

RPB must fashion tools from two ADFs which only have recourse to the operations of arithmetic. Specifically, the automatically defined functions were composed of {+, -, *, %} operating on {yhotSpot, yhotSpot+1, yhotSpot-1} and a random constant chosen from {1.0, 2.0, ..., 15.0}, an arbitrary set. The problem size was increased in terms of population, generations, and tree depth, in keeping with the experience gained from the earlier, analogous case using BUMP and GRIND; see Table 2.

While the computer's operating system was tolerant of infinities and undefined floating point operations, the arithmetic operations were protected for the sake of monitoring statistics such as mean and variance of the population's fitness values. Thus % was ordinary division unless the result would exceed $1.0e+15$ in magnitude, in which case the value returned was 1.0, consistent with the spirit of protected division in (Koza 1992). The other operations were simply capped at $\pm 1.0e+15$, and standardized fitness was capped at $1.0e+30$. This mild protection permitted wide (perhaps excessive) variation in the magnitude of the shape alterations.

Each time the result producing branch called one of the ADFs, the y-value at the current hotSpot was replaced by the quantity calculated within the ADF—this was implemented as a side-effect. The intention was to mimic the earlier case, but with evolvable ADFs in place of BUMP and GRIND (which are expressible in the same terms as were available to the ADFs). This would ensure the existence of solutions at least as good as those reported previously.

This attempt was not fully successful. Of 10 runs, lasting on average four hours apiece, only two made any progress whatsoever. The rest immediately hit upon one of a variety of ways to leave the initial shape unchanged, and never progressed. Of course, given that standard fitness could, and did, reach as high as $1.0e+30$, the "do nothing" score of 0.429204 does not look so bad. A noteworthy difference between these and previous runs was that while there was wide variation in the initial, random population's fitnesses, the variation was invariably on the side of worse fitness than the undisturbed embryo's—this despite a population

Table 3. GP tableau for the diamond-shaped airfoil problem using an IPB, new TAP operation, and increased population size and depth limit for new trees.

Objective	Minimize drag subject to thickness constraint.
Terminals	RPB: ADF, CONST (values from 1 to 15) ADF: INC, DEC, BUMP, GRIND, TAP IPB: ADF, INC, DEC, BUMP, GRIND, TAP
Functions	RPB: IPB, PROGN (2-branch) ADF: PROGN IPB: PROGN
Standardized Fitness	(Drag - 0.040) + 1,000 max (0.10 - Thickness, 0)
Success Predicate	Standardized Fitness \leq 0.0040
Parameters	Population 10,000 Max Nodes Per RPB 200 Generations 31 Max Depth New Tree 6

Examination of the "near-miss" shapes revealed a common feature: most were too blunt fore and aft, but matched the known solution in the middle of the chord. Since it seemed to be difficult to *reduce* the thickness at the ends of the shape-defining array (perhaps because the leading and trailing edge points were fixed), a new elementary operator was defined. TAP, described above, reduces thickness by a fixed, absolute amount scaled to the problem, and turned out to be what was needed. Another set of parameter-varying runs resulted in the formulation summarized in Table 3. Note that both the population size and maximum new tree depth have been increased. In return, for three runs out of three, success is achieved in an average of 22 generations. The time required averaged about two hours per run to meet the termination criterion.

Among the better results observed, on one run a standard fitness of 0.002458 was reached on generation 18. The 157-node program which produced the result is reproduced below, and is no more comprehensible than the previous examples. The iteration performing branch is highlighted in boldface type. The effect of just a single iteration on the starting airfoil gives some idea of what is happening. This is plotted to exaggerated scale in Figure 2, along with the theoretical optimum shape, the initial airfoil, and the final result. The IPB uses all the available terminals and functions to alter the starting shape in a plausible, but non-intuitive way.

Run 355 Gen:18 BestFitness:0.00246

RPB_Num 0 (31 nodes)

```
(PROGN (PROGN (PROGN (PROGN (ADF0) 3.0)
  (PROGN (ADF0) (ADF0))) (ADF1 (PROGN (PROGN
    (ADF0) (ADF0)) (PROGN (ADF0) (ADF0))))))
  (PROGN (ADF1 (ADF1 (ADF1 2.0))) (ADF1
    (ADF1 (PROGN (ADF1 (ADF1 (ADF0))) (ADF1
      (ADF1 2.0)))))))
```

ADF_Num 0 (71 nodes)

```
(PROGN (PROGN (PROGN (PROGN (PROGN
  (BUMP) (INC)) (PROGN (TAP) (BUMP)))
```

```
(PROGN (PROGN (BUMP) (GRIND)) (PROGN
  (INC) (BUMP))) (PROGN (PROGN (PROGN
    (BUMP) (DEC)) (PROGN (DEC) (BUMP))) (PROGN
      (PROGN (DEC) (TAP)) (PROGN (TAP) (GRIND))))))
  (PROGN (PROGN (PROGN (PROGN (GRIND)
    (TAP)) (PROGN (TAP) (BUMP))) (PROGN (PROGN
      (INC) (BUMP)) (PROGN (BUMP) (TAP)))) (PROGN
        (PROGN (PROGN (INC) (GRIND)) (PROGN
          (GRIND) (DEC))) (PROGN (PROGN (INC) (BUMP))
            (PROGN (PROGN (INC) (BUMP)) (PROGN (PROGN
              (BUMP) (DEC)) (PROGN (BUMP) (INC)))))))
```

ADF_Num 1 (55 nodes)

```
(PROGN (PROGN (PROGN (PROGN (PROGN
  (ADF0) (GRIND)) (PROGN (ADF0) (ADF0)))
  (PROGN (PROGN (DEC) (DEC)) (PROGN
    (ADF0) (ADF0)))) (PROGN (PROGN
      (ARG0) (ADF0)) (PROGN (PROGN (PROGN
        (BUMP) (BUMP)) (PROGN (GRIND) (GRIND)))
        (PROGN (PROGN (ADF0) (ADF0)) (PROGN
          (TAP) (BUMP)))))) (PROGN (PROGN (PROGN
            (BUMP) (BUMP)) (PROGN (PROGN (TAP) (DEC))
              (PROGN (DEC) (BUMP)))) (PROGN (PROGN
                (BUMP) (INC)) (PROGN (BUMP) (BUMP))))))
```

The shape crafted by this program is a close match to the known optimum. The thickness constraint is active, and the expected fore/aft symmetry has appeared. This solution required approximately 220,000 fitness evaluations, though this is probably not be the best that can be achieved.

6. Future Work

These test problems were simple enough to solve by other, more conventional (and efficient) means. Indeed, with the static parameterization and modest number of variables employed here, straightforward application of the genetic algorithm on the $\{y_i\}$ would presumably work, and without all the overhead required by "navigation" within the array. It will be interesting to see whether there are non-trivial shaping problems susceptible to solution by GP. In

Table 4. Summary of semicircle problem outcomes.

Description	Runs	Success	Average No. Gens.	Average Duration	Parameters
RPB with Two Specialized ADFs	25	36%			Table 1
RPB with No ADFs	10	0%			Table 1
	3	100%	11	20 min	Larger pop., depth, nodes
RPB with ADF and IPB	20	100%	10	3.5 min	Table 1
RPB with Two ADFs; "Hands Off"	10	0%			Larger pop., depth, nodes, gens.

The approach employed, using the hotSpot pointer and navigation operators on the array representing the shape, and using side effects to change the shape, proved to work reasonably well. But the present test problems did not explore whether this approach can add enough flexibility to compensate for the navigational overhead; the number and location of the shape's defining points remained fixed, for example.

Automatically defined functions, especially those capable of iteration, provide powerful leverage in obtaining a solution. ADFs proved useful despite the lack of any obvious exploitable regularity in the problems—this may be a new result. The solutions found with ADFs, especially those capable of iteration, were much smaller than those without; indeed, in the latter case the semicircle problem was not solvable until the node limit was raised from 200 to 1,000. Thus the subtle regularities discovered using ADFs and iteration were evidently sufficient to fit the necessary number of operations within the node budget, and these solutions were also obtained more quickly.

No special parameterization of the problem was required, but selection of a sufficiently-general set of elementary operations appears to take its place as an important preliminary step. Perhaps a universal set can be formulated or evolved. In any event, the intentionally-naïve set employed here, and the initial difficulties with tool evolution reported, should not be taken as the last word on function selection.

Substantial computer effort was required even for the test problems' simple fitness functions. Some additional work will be required to determine whether better choice of elementary operations, further structural refinements, etc., are able to improve efficiency significantly. But since flow analysis codes used in current design optimization require from 0.10 sec to 1,000 sec per evaluation, large-scale parallel computers will probably be required for real-life applications of genetic programming to aerodynamics.

References

- Grasmeyer, Joel. 1997. Application of a Genetic Algorithm with Adaptive Penalty Functions to Airfoil Design. AIAA 97-0007. January 6–10, Reno, NV.
- Hague, D. S., H. L. Rozendaal, and F. A. Woodward. 1968. Application of Multivariable Search Techniques to Optimal Aerodynamic Shaping Problems. *J. Astronaut. Sci.* 15:283–296.
- Hicks, R. M., E. M. Murman, and G. N. Vanderplaats. 1974. An Assessment of Airfoil Design by Numerical Optimization. NASA TM X-3092.
- Holland, John H. 1992. *Adaptation in Natural and Artificial Systems*. 2d ed. Cambridge, MA: MIT Press.
- Kennelly, Jr., Robert A. 1983. Improved Method for Transonic Airfoil Design-by-Optimization. AIAA 83-1864. July 13–15, Danvers, MA.
- Kennelly, Jr., Robert A. 1997. Genetic Evolution of Shape-Altering Programs for Supersonic Aerodynamics. In Koza, John R. (compiler). *Genetic Algorithms and Genetic Programming at Stanford 1997*. Stanford, CA: Stanford University Bookstore.
- Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.
- Liepmann, Hans Wolfgang, and Allen E. Puckett. 1947. *Introduction to Aerodynamics of a Compressible Fluid*. New York: John Wiley & Sons. Chapter 9.
- Quagliarella, Domenico, and Antonio Della Cioppa. 1995. Genetic Algorithms Applied to the Aerodynamic Design of Transonic Airfoils. *J. Aircraft* 32:889–891.
- Rubbert, Paul E. 1994. CFD and the Changing World of Airplane Design. AIAA Wright Brothers Lecture. Sept. 18–23, Anaheim, CA.

